Parallel implementation of FRAME – Filters, Random Fields and Maximum Entropy

Ricardo N. Rodrigues
rnr4@buffalo.edu
CSE633-Parallel Algorithms
University at Buffalo

• Texture modeling:

Figure 10. (a) The observed texture—cheetah blob, and (b) the synthesized one using six filters.
Method Overview

Input Image: \( I^{\text{obs}} \)

Filters: \( S_K = \{ F^{(1)}, \ldots, F^{(K)} \} \)

Histograms:
\[ \{ H^{\text{obs}(\alpha)}, \quad \alpha = 1, \ldots, K \}. \]

Model:
\[
p(I; \Lambda_K, S_K)
\]
\[
= \frac{1}{Z(\Lambda_K)} \exp \left\{ - \sum_{\alpha=1}^{K} \sum_{i=1}^{L} \lambda^{(\alpha)}_i H^{(\alpha)}_i \right\},
\]
\[
= \frac{1}{Z(\Lambda_K)} \exp \left\{ - \sum_{\alpha=1}^{K} \langle \lambda^{(\alpha)}, H^{(\alpha)} \rangle \right\}.
\]
Algorithm 1. The FRAME Algorithm

Input a texture image $I^{\text{obs}}$.
Select a group of $K$ filters $S_K = \{F^{(1)}, F^{(2)}, \ldots, F^{(K)}\}$.
Compute $\{H^{\text{obs}(\alpha)}, \alpha = 1, \ldots, K\}$.
Initialize $\lambda^{(\alpha)}_i \leftarrow 0, \quad i = 1, 2, \ldots, L, \quad \alpha = 1, 2, \ldots, K$.
Initialize $I^{\text{syn}}$ as a uniform white noise texture.
Repeat
  Calculate $H^{\text{syn}(\alpha)} \alpha = 1, 2, \ldots, K$ from $I^{\text{syn}}$, use it for $E_{p(I; \Lambda_K, S_K)}(H^{(\alpha)})$.
Update $\lambda^{(\alpha)}_i \alpha = 1, 2, \ldots, K$ by Eq. (19), $p(I; \Lambda_K, S_K)$ is updated.
Apply Gibbs sampler to flip $I^{\text{syn}}$ for $w$ sweeps under $p(I; \Lambda_K, S_K)$
Until $\frac{1}{2} \sum_i^L |H^{\text{obs}(\alpha)}_i - H^{\text{syn}(\alpha)}_i| \leq \epsilon$ for $\alpha = 1, 2, \ldots, K$.

Algorithm 2. The Gibbs Sampler for $w$ Sweeps

Given image $I(\bar{v})$, flip_counter $\leftarrow 0$
Repeat
  Randomly pick a location $\bar{v}$ under the uniform distribution.
  For $\text{val} = 0, \ldots, G - 1$ with $G$ being the number of grey levels of $I$
    Calculate $p(I(\bar{v}) = \text{val} | I(-\bar{v}))$ by $p(I; \Lambda_K, S_K)$.
    Randomly flip $I(\bar{v}) \leftarrow \text{val}$ under $p(\text{val} | I(-\bar{v}))$.
    flip_counter $\leftarrow$ flip_counter + 1
Until flip_counter $= w \times M \times N$. 
Implementation Issues

• Algorithm 2 (Gibbs sampler) was parallelized.
  – Work shared in the “filter level”
  – So far I have used 24 filters

• Parallelized for shared memory machines using OpenMP

• Parallel region -> 87.5%

• MKL optimizations not running yet due to different versions of MKL in U2 and LENNON.
Running time for different image size

![Graph showing running time for different image sizes.](image-url)
Speed up

![Graph showing speed up vs. processors with lines for 128, 256, and Amdahl 0.875.]
Serial code profiling using Intel VTune

![Sampling Hotspots - [Run 1]](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>CPU_sample</th>
<th>INST_R samples</th>
<th>Clocks_per_1000M</th>
<th>CPU_CL %</th>
<th>INST_R %</th>
<th>CPU_CLK_UNHalted</th>
<th>INST_RETIRED.A</th>
<th>Se</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_histogram</td>
<td>419,691</td>
<td>291,960</td>
<td>1.437</td>
<td>37.12%</td>
<td>22.59%</td>
<td>895,200,903,000</td>
<td>622,750,680,000</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>sampleGivenNatParam</td>
<td>376,865</td>
<td>641,451</td>
<td>0.588</td>
<td>33.33%</td>
<td>49.64%</td>
<td>803,831,715,000</td>
<td>1,368,214,983,000</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>ftoi2_pentium4</td>
<td>119,095</td>
<td>65,102</td>
<td>1.829</td>
<td>10.53%</td>
<td>5.04%</td>
<td>254,029,635,000</td>
<td>138,862,566,000</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>memcpy</td>
<td>43,575</td>
<td>14,840</td>
<td>2.936</td>
<td>3.85%</td>
<td>1.15%</td>
<td>92,945,475,000</td>
<td>31,653,720,000</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>ftoi2_sse</td>
<td>34,463</td>
<td>108,885</td>
<td>0.317</td>
<td>3.05%</td>
<td>8.43%</td>
<td>73,509,579,000</td>
<td>232,251,705,000</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>assign&lt;float&gt;</td>
<td>22,149</td>
<td>50,404</td>
<td>0.439</td>
<td>1.96%</td>
<td>3.90%</td>
<td>47,243,817,000</td>
<td>107,511,732,000</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>mkl_blas_p4m_dgemv</td>
<td>16,675</td>
<td>20,053</td>
<td>0.832</td>
<td>1.47%</td>
<td>1.55%</td>
<td>35,567,775,000</td>
<td>42,773,049,000</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>fastcopy_l</td>
<td>11,940</td>
<td>11,551</td>
<td>1.034</td>
<td>1.06%</td>
<td>0.89%</td>
<td>25,468,020,000</td>
<td>24,638,283,000</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>VEC_memzero</td>
<td>8,478</td>
<td>5,878</td>
<td>1.442</td>
<td>0.75%</td>
<td>0.45%</td>
<td>18,083,574,000</td>
<td>12,537,774,000</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>VEC_memcpy</td>
<td>7,653</td>
<td>3,323</td>
<td>2.303</td>
<td>0.68%</td>
<td>0.26%</td>
<td>16,323,849,000</td>
<td>7,087,959,000</td>
<td>0xFFFF</td>
</tr>
</tbody>
</table>
Still working on…

- Extend the use of MKL for more optimization
- Test with different input configurations.